The team orienteering problem with overlaps

Orlis, Christos; Bianchessi, Nicola; Roberti, Roberto; Dullaert, Wout

published in
Transportation Science
2020

DOI (link to publisher)
10.1287/trsc.2019.0923

document version
Publisher's PDF, also known as Version of record

citation for published version (APA)
Orlis, C., Bianchessi, N., Roberti, R., & Dullaert, W. (2020). The team orienteering problem with overlaps: An application in cash logistics. *Transportation Science*, *54*(2), 470-487. https://doi.org/10.1287/trsc.2019.0923

Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

The Team Orienteering Problem with Overlaps: An Application in Cash Logistics

Christos Orlis, Nicola Bianchessi, Roberto Roberti, Wout Dullaert

To cite this article:
Christos Orlis, Nicola Bianchessi, Roberto Roberti, Wout Dullaert (2020) The Team Orienteering Problem with Overlaps: An Application in Cash Logistics. Transportation Science 54(2):470-487. https://doi.org/10.1287/trsc.2019.0923

http://pubsonline.informs.org/journal/trsc

TRANSPORTATION SCIENCE
Vol. 54, No. 2, March–April 2020, pp. 470–487
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

The Team Orienteering Problem with Overlaps: An Application in Cash Logistics

Christos Orlis,[a] Nicola Bianchessi,[a,b] Roberto Roberti,[a] Wout Dullaert[a]

[a] Department of Supply Chain Analytics, Vrije Universiteit Amsterdam, 1081 HV Amsterdam, Netherlands; [b] Dipartimento di Informatica, Università degli Studi di Milano, 20122 Milan, Italy

Contact: c.orlis@vu.nl (CO); bianchessi@di.unimi.it, https://orcid.org/0000-0002-5722-5476 (NB); r.roberti@vu.nl, https://orcid.org/0000-0002-2987-1593 (RR); w.e.h.dullaert@vu.nl (WD)

Received: November 28, 2018
Revised: May 17, 2019
Accepted: June 6, 2019
Published Online in Articles in Advance: March 4, 2020

https://doi.org/10.1287/trsc.2019.0923

Abstract. The *team orienteering problem* (*TOP*) aims at finding a set of routes subject to maximum route duration constraints that maximize the total collected profit from a set of customers. Motivated by a real-life automated teller machine cash replenishment problem that seeks for routes maximizing the number of bank account holders having access to cash withdrawal, we investigate a generalization of the TOP that we call the *team orienteering problem with overlaps* (*TOPO*). For this problem, the sum of individual profits may overestimate the real profit. We present exact solution methods based on column generation and a metaheuristic based on large neighborhood search to solve the TOPO. An extensive computational analysis shows that the proposed solution methods can efficiently solve synthetic and real-life TOPO instances. Moreover, the proposed methods are competitive with the best algorithms from the literature for the TOP. In particular, the exact methods can find the optimal solution of 371 of the 387 benchmark TOP instances, 33 of which are closed for the first time.

Funding: This work was supported by the Dutch Science Foundation [Grant 407-13-050].
Supplemental Material: The e-companion is available at https://doi.org/10.1287/trsc.2019.0923.

Keywords: team orienteering • cash distribution • routing with profits • column generation • metaheuristic

1. Introduction

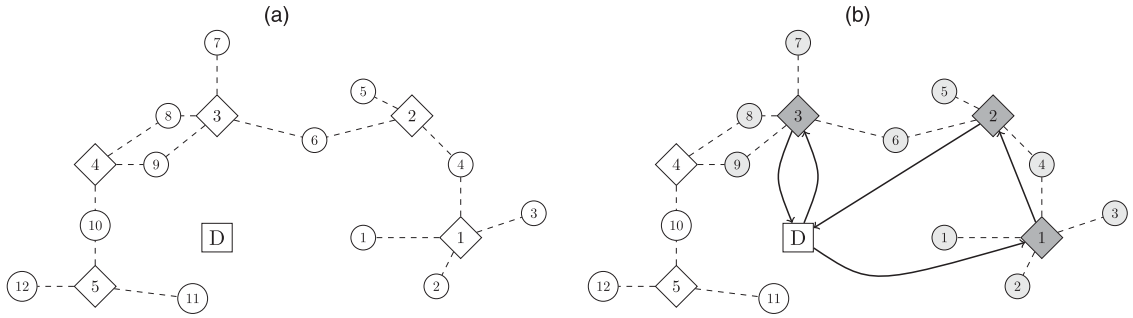
The *team orienteering problem* (TOP) is a well-known decision-making problem of the class of the vehicle routing problem with profits (see Archetti, Speranza, and Vigo 2014). Given a set of vehicles and a set of customers, each one with an associated profit, the goal of the TOP is to design a set of routes (one for each vehicle) that maximizes the total profit collected by visiting (some of) the customers without exceeding a maximum duration constraint for each route.

The first paper on the TOP was by Butt and Cavalier (1994). Since then, many exact and heuristic approaches to solve the TOP and many of its variants have appeared in the literature (for recent surveys on the topic, see Vansteenwegen, Souffriau, and Van Oudheusden 2011; Gunawan, Lau, and Vansteenwegen 2016). The increasing interest in efficient solution methods to solve the TOP and related problems is due to the variety of real-life applications that can be modeled as TOPs, for example, athlete recruiting (Chao, Golden, and Wasil 1996), technician routing (Tang and Miller-Hooks 2005), design of tourist trips (Vansteenwegen and Van Oudheusden 2007), and customer selection in less-than-truckload transportation (Archetti et al. 2009).

In this paper, we introduce and solve a new generalization of the TOP that we call the *team orienteering problem with overlaps* (*TOPO*). The TOPO is inspired by a real-life decision-making problem faced by Geldmaat, a joint venture of the three largest Dutch commercial banks (ABN AMRO, ING, and Rabobank) in charge of providing logistical services such as cash collection, counting, and distribution in the Netherlands. Automated teller machines (ATMs) are replenished by using a set of armored vehicles, whose routes are subject to maximum duration constraints deriving from legal restrictions that limit the time each driver can work every day. One of the main challenges faced by Geldmaat is to decide which ATMs to replenish to maximize the number of bank account holders that have access to ATMs. This is particularly challenging on days with peak demand, when most ATMs are empty and vehicles are in short supply. In our real-life setting, bank account holders (identified by the postal code of residence) are considered to have access to cash if there exists a replenished ATM within five kilometers from the residence postal code.

Figure 1(a) shows an example of a TOPO instance with five ATMs and 12 bank account holders. The depot is represented by the rectangle (D), the ATMs

Figure 1. Example of (a) a TOPO Instance and (b) a Corresponding Feasible Solution



by diamonds (1–5), and the bank account holders by circles (numbered from 1 to 12). A dashed line between an ATM and a bank account holder indicates that the ATM can serve the bank account holder. Some bank account holders (e.g., 4, 6, 8, 9, and 10) can be served by two ATMs. A feasible solution for the instance is depicted in Figure 1(b). Two vehicles are used. The first serves ATMs 1 and 2, whereas the other serves just ATM 3. This solution serves nine bank account holders (i.e., the grey circles). Notice that bank account holders 4 and 6 can withdraw from both ATMs, but they do not count twice in the total number of bank account holders served.

The TOPO also arises in other real-life distribution problems. A first application is the decision on which stores to replenish given that consumers can be served from one or several nearby stores. Another application is in humanitarian logistics, where first-aid resources or services have to be provided in large geographical areas and beneficiaries can be served at different service points. As the TOPO does not model just the problem faced by Geldmaat but also other real-life applications, we will refer to the ATMs as *service points* and the bank account holders as *consumers* in the rest of this paper.

The main contributions of this paper are the following: (i) we introduce the TOPO, a new generalization of the TOP that arises in real-life applications, together with (ii) exact branch-and-cut-and-price (BCP) algorithms that can solve to optimality instances with up to 100 service points; (iii) we show how the *ng*-path relaxation technique, introduced by Baldacci, Mingozzi, and Roberti (2011) to price out columns associated with possibly nonelementary paths, can be exploited not only to prevent some cycles but also to improve the linear relaxation bound by lifting some of the coefficients of the columns priced out; (iv) we propose a *large neighborhood search* (*LNS*) metaheuristic for the TOPO that is able to find high-quality solutions of instances with up to 100 service points within short computation times; (v) we report on experimental results obtained by applying the best performing of our BCP algorithms to solve the TOP benchmark instances by Chao, Golden, and Wasil (1996), showing that it improves upon state-of-the-art exact methods by solving to optimality about 96% of the instances (371 out of 387) and closing 33 open instances; (vi) we show that both the best-performing BCP algorithm and the LNS metaheuristic can provide high-quality solutions for real-life instances provided by Geldmaat with up to 100 service points and 203,717 consumers; (vii) from a managerial point of view, we report a quantitative analysis that sheds light on the potential losses that arise when the underlying real-life problem should be modeled as a TOPO but is addressed as a TOP.

The rest of this paper is organized as follows. Section 2 reviews the main contributions from the literature on exact and heuristic algorithms for the TOP. The TOPO is formally introduced in Section 3, where a compact formulation for the problem is provided. Section 4 describes the BCP algorithms. Section 5 illustrates the LNS metaheuristic. Section 6 discusses the computational results obtained by applying the devised algorithms on both synthetic and real-life instances. Finally, conclusions and future research directions are outlined in Section 7.

2. Literature Review

As the TOPO generalizes the well-known TOP (see Section 3), this section reviews the main exact and heuristic methods presented in the literature to solve the TOP. For an exhaustive overview of the literature on the TOP and related problems, the reader is referred to the surveys of Vansteenwegen, Souffriau, and Van Oudheusden (2011) and Gunawan, Lau, and Vansteenwegen (2016).

2.1. Exact Methods

The first exact method for the TOP is owed to Boussier, Feillet, and Gendreau (2007), who describe a branch-and-price algorithm based on a set packing formulation where each variable represents an elementary route. The pricing problem corresponds to an elementary shortest path problem with resource constraints (ESPPRC). Different acceleration techniques are proposed to solve the ESPPRC. The proposed branch-and-price is tested on the standard benchmark instances introduced by Chao, Golden, and Wasil (1996),

hereafter referred to simply as *Chao instances*. The computational results show that 270 of the 387 Chao instances can be solved to optimality within two hours of computational time.

Poggi, Viana, and Uchoa (2010) describe three mathematical formulations and a robust branch-and-cut-and-price algorithm, where the pricing problem is solved by dynamic programming and two families of robust cuts (i.e., min cuts and triangle clique cuts) are separated. A partial implementation of the branching schemes is adopted, so only partial preliminary computational results on the Chao instances are reported.

Dang, El-Hajj, and Moukrim (2013) propose a branch-and-cut method based on a three-index formulation with a polynomial number of binary variables. The linear relaxation is tightened by adding different sets of valid inequalities and dominance properties, such as boundaries on profits, symmetry breaking, generalized subtour elimination, and clique constraints. The proposed branch-and-cut can solve 278 Chao instances within two hours of computational time.

Keshtkaran et al. (2016) build on the exact method of Boussier, Feillet, and Gendreau (2007) to propose an enhanced branch-and-cut-and-price algorithm, where the pricing problem is solved by bounded bidirectional dynamic programming with decremental state-space relaxation and two-phase dominance rule relaxation. The subset-row (SR) inequalities introduced by Jepsen et al. (2008) are also separated to strengthen the linear relaxation of the set packing formulation. The proposed branch-and-cut-and-price can solve 301 Chao instances within two hours of computational time.

El-Hajj, Dang, and Moukrim (2016) describe a branch-and-cut algorithm building on the three-index formulation and the cuts used by Dang, El-Hajj, and Moukrim (2013), but based on additional sets of valid inequalities. The resulting branch-and-cut approach can solve 300 Chao instances within two hours of computational time.

The most recent exact method is owed to Bianchessi, Mansini, and Speranza (2018), who introduce a branch-and-cut method based on a new two-index formulation with a polynomial number of variables and constraints. The linear relaxation is enforced by separating an exponential number of connectivity constraints with an exact algorithm for max-flow/min-cut problems. The proposed branch-and-cut algorithm can solve 327 of the 387 Chao instances within two hours of computational time. Nevertheless, 49 of the 387 Chao instances have not been solved to optimality by any of the exact methods presented so far in the literature.

2.2. Heuristic Methods

A variety of heuristics and metaheuristics have been proposed for the TOP since the early 2000s. In general, neighborhood-based approaches are more common than population-based approaches.

2.2.1. Neighborhood-Based Approaches.

Tang and Miller-Hooks (2005) introduce a tabu search heuristic embedded in an adaptive memory procedure that explores both feasible and infeasible solutions, alternates between small and large neighborhoods in the solution improvement phase, and uses greedy and random components for generating neighborhood solutions.

Archetti, Hertz, and Speranza (2007) describe three metaheuristics, that is, two generalized tabu search algorithms and a variable neighborhood search algorithm. The algorithms explore feasible and infeasible solutions. The impact of different strategies to jump between solutions, penalize infeasibility, and restore feasibility is assessed.

Vansteenwegen et al. (2009) describe an algorithm that combines different local search heuristics and uses guided local search to improve the quality of the solutions achieved.

Two versions of a metaheuristic based on a pure path relinking approach combined with a greedy randomized adaptive search procedure are presented in Souffriau et al. (2010).

Lin (2013) proposes a multistart simulated annealing algorithm that combines simulated annealing with multistart hill climbing to minimize the chances of being trapped in local minima.

Kim, Li, and Johnson (2013) propose a large neighborhood search embedding three improvement algorithms (i.e., a local search, a shift-and-insertion, and a replacement improvement) and manage to compute all 387 best-known solutions of the Chao Instances.

Vidal et al. (2015) introduce a heuristic framework for solving the TOP and two other vehicle routing problems with profits and manage to compute the best-known solutions for all but one of the Chao instances. The heuristic is based on an exhaustive solution representation where first all customers are assigned to the vehicles and in a sequence that possibly violates the maximum-route duration constraints. Then, the final set of customers to be served is selected by exploring, in pseudopolynomial time, an exponential neighborhood of solutions. Notice that the approach of Vidal et al. (2015) could also be classified as a population-based approach.

2.2.2. Population-Based Approaches.

An ant colony optimization approach is proposed by Ke, Archetti, and Feng (2008). Four alternative methods (i.e., sequential, deterministic-concurrent, random-concurrent, and simultaneous) are applied to construct candidate solutions within the ant colony framework.

Bouly, Dang, and Moukrim (2010) present a memetic algorithm that combines genetic algorithms with

local search techniques to improve the mutation phase. The encoding of a solution is based on a giant-tour representation, and an optimal split procedure is applied to evaluate the chromosomes. The algorithm of Bouly, Dang, and Moukrim (2010) was later hybridized with particle-swarm optimization by Dang, Guibadj, and Moukrim (2011) and further extended by Dang, Guibadj, and Moukrim (2013), who managed to compute all 387 best-known solutions of the Chao instances.

Ke et al. (2016) present a metaheuristic using a Pareto-dominance criterion to control the similarity between the generated and the incumbent solutions. Pareto dominance is also used to update the population of incumbent solutions. This metaheuristic, similarly to those of Kim, Li, and Johnson (2013) and Dang, Guibadj, and Moukrim (2013), manages to compute all 387 best-known solutions for the Chao instances.

3. Problem Description

The TOPO can be formally described as follows. A set of consumers $\mathscr{C}$ is given. Consumers are served via a set of service points $\mathscr{S} = \{1, 2, \ldots, n\}$; in particular, each consumer $c \in \mathscr{C}$ can be served by a subset of service points $\mathscr{S}_c \subseteq \mathscr{S}$. Similarly, the subset of consumers that can be served by service point $i \in \mathscr{S}$ is indicated by $\mathscr{C}_i \subseteq \mathscr{C}$. Consumers can be served via the service points by routing a set of homogeneous vehicles $\mathscr{K}$ located at a depot, indicated by 0. Each vehicle can perform a route that starts from the depot, visits some service points, and returns to the depot. Each route cannot exceed a maximum route duration denoted by T. The travel time between each pair of depot/service point locations i and j ($i, j \in \mathscr{V} = \mathscr{S} \cup \{0\}$) is indicated by t_{ij}. Travel times can be asymmetric and, without loss of generality, are assumed to be strictly positive and to satisfy the triangle inequality (i.e., $t_{ij} \leq t_{ik} + t_{kj}$ for each $i, j, k \in \mathscr{V}$). The TOPO aims at finding a set of routes, each one not exceeding the maximum route duration, that visit each service point at most once and maximize the number of consumers served.

In the TOP, a set of customers $\mathscr{U}$ is given. A profit r_u is associated with each customer $u \in \mathscr{U}$, and a set of m vehicles is located at the depot and can be used to serve the customers. The goal of the TOP is to find a set of at most m routes, each one not exceeding a maximum route duration T, such that each customer is served at most once and the total collected profit is maximized. We can observe that the TOP is a special case of the TOPO. Indeed, any TOP instance can be mapped into a TOPO instance as follows. The set $\mathscr{K}$ comprises m homogeneous vehicles; that is, $|\mathscr{K}| = m$. Each customer of the TOP instance corresponds to a service point in the TOPO instance, and the TOPO instance contains $|\mathscr{C}| = \sum_{u \in \mathscr{U}} r_u$ consumers. The sets $\mathscr{C}_u$ are defined in such a way that $\mathscr{C}_u$, $u \in \mathscr{U}$, contains r_u consumers (i.e., $|\mathscr{C}_u| = r_u$), and the sets $\mathscr{C}_u$ are pairwise disjoint (i.e., $\mathscr{C}_u \cap \mathscr{C}_{u'} = \emptyset,\ u, u' \in \mathscr{U} : u \neq u'$). Moreover, the set $\mathscr{S}_c$, $c \in \mathscr{C}$, contains a single element that corresponds to the only service point u such that $c \in \mathscr{C}_u$. It is easy to observe that any solution of the resulting TOPO instance corresponds to a solution of the original TOP instance.

The TOPO can be defined on a directed graph $\mathscr{G} = (\mathscr{V}, \mathscr{A})$, where the arc set is defined as $\mathscr{A} = \{(i, j) \mid i, j \in \mathscr{V} : i \neq j\}$. Let us define the following three sets of variables: $x_{ij} \in \{0, 1\}$, a binary variable equal to 1 if arc $(i, j) \in \mathscr{A}$ is traversed by one of the vehicles (0 otherwise); $y_c \in \{0, 1\}$, a binary variable equal to 1 if consumer $c \in \mathscr{C}$ is served (0 otherwise); and $z_i \in \mathbb{R}_+$, a continuous variable indicating the arrival time at service point $i \in \mathscr{S}$. Then, the TOPO can be formulated as follows:

$$z^* = \max \sum_{c \in \mathscr{C}} y_c \tag{1a}$$

$$\text{s.t.} \sum_{(0,j) \in \mathscr{A}} x_{0j} \leq |\mathscr{K}|, \tag{1b}$$

$$\sum_{(i,j) \in \mathscr{A}} x_{ij} \leq 1 \qquad i \in \mathscr{S}, \tag{1c}$$

$$\sum_{(i,j) \in \mathscr{A}} x_{ij} = \sum_{(j,i) \in \mathscr{A}} x_{ji} \qquad i \in \mathscr{S}, \tag{1d}$$

$$z_i + (T + t_{ij}) x_{ij} \leq z_j + T \quad (i, j) \in \mathscr{A} : i, j \in \mathscr{S}, \tag{1e}$$

$$t_{0i} \sum_{(i,j) \in \mathscr{A}} x_{ij} \leq z_i \leq (T - t_{i0}) \sum_{(i,j) \in \mathscr{A}} x_{ij} \qquad i \in \mathscr{S}, \tag{1f}$$

$$\sum_{(i,j) \in \mathscr{A} : i \in \mathscr{S}_c} x_{ij} \geq y_c \qquad c \in \mathscr{C}, \tag{1g}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j) \in \mathscr{A}, \tag{1h}$$

$$y_c \in \{0, 1\} \qquad c \in \mathscr{C}. \tag{1i}$$

The objective function (1a) asks for maximizing the number of consumers served. Constraint (1b) ensures that no more than $|\mathscr{K}|$ routes are designed. Constraints (1c) guarantee that each service point is visited at most once. Constraints (1d) are flow conservation constraints for the service points. Constraints (1e) link x and z variables to set the arrival time at each service point based on the traversed arcs and also prevent subtours in the designed routes. Constraints (1f) guarantee that if service point $i \in \mathscr{S}$ is visited, the arrival time of the vehicle visiting it is not less than the travel time from the depot to i and not greater than $T - t_{i0}$. Constraints (1g) ensure that each consumer $c \in \mathscr{C}$ is served only if at least one of the service points of the

set $\mathcal{S}_c$ is visited. Constraints (1h) and (1i) define the range of the decision variables.

4. Exact Methods

In this section, we discuss the exact branch-and-cut-and-price algorithms devised to address the TOPO. In Section 4.1, we introduce a first BCP algorithm. Then, starting from the BCP algorithm just presented, two alternative, potentially improving versions are derived and discussed in Section 4.2.

4.1. Branch-and-Cut-and-Price Algorithm

In the following, we present the main features and components of the BCP algorithm.

4.1.1. Route-Based Model. Let $\mathbb{H} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_{|\mathbb{H}|}\}$ be a partition of the consumers such that $\mathcal{S}_c = \mathcal{S}_{c'}$ for each pair of consumers $c, c' \in \mathcal{C}$, $c \neq c'$, belonging to the same subset $\mathcal{H}_h$; that is, the two consumers can be served by the same subset of service points. In the following, each subset of the partition $\mathbb{H}$ is called *class of consumers* (or simply *class*), and we refer to each class corresponding to subset $\mathcal{H}_h$ by its index h. Moreover, let $\mathcal{S}(\mathcal{H}_h) \subseteq \mathcal{S}$ be the subset of service points that serve the consumers of the subset $\mathcal{H}_h$, that is, $\mathcal{S}(\mathcal{H}_h) = \mathcal{S}_c$ for each $c \in \mathcal{H}_h$, and let $\mathcal{H}(i)$ be the subset of classes served by service point $i \in \mathcal{S}$, that is, $\mathcal{H}(i) = \{h \,|\, h = 1, \ldots, |\mathbb{H}|, i \in \mathcal{S}(\mathcal{H}_h)\}$. Let $\mathcal{R}$ be the set of all feasible routes in graph $\mathcal{G}$, and let a_{ir} be a integer coefficient indicating the number of times route $r \in \mathcal{R}$ visits service point $i \in \mathcal{S}$.

Let $\delta_h \in \mathbb{R}_+$ be a nonnegative continuous variable (with a binary meaning) equal to 1 if the class of consumers $h = 1, \ldots, |\mathbb{H}|$ is not served (0 otherwise), and let $\xi_r \in \{0, 1\}$ be a binary variable equal to 1 if route $r \in \mathcal{R}$ is selected (0 otherwise). The TOPO can be formulated as follows:

$$[\overline{\mathrm{F}}] \quad \delta^* = \min \sum_{h=1}^{|\mathbb{H}|} |\mathcal{H}_h| \delta_h \tag{2a}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} a_{ir} \xi_r \leq 1 \qquad i \in \mathcal{S}, \tag{2b}$$

$$\sum_{r \in \mathcal{R}} \xi_r \leq |\mathcal{K}|, \tag{2c}$$

$$\sum_{r \in \mathcal{R}} \left(\sum_{i \in \mathcal{S}(\mathcal{H}_h)} a_{ir} \right) \xi_r + \delta_h \geq 1 \quad h = 1, \ldots, |\mathbb{H}|, \tag{2d}$$

$$\xi_r \in \{0, 1\} \qquad r \in \mathcal{R}, \tag{2e}$$

$$\delta_h \in \mathbb{R}_+ \qquad h = 1, \ldots, |\mathbb{H}|. \tag{2f}$$

The objective function (2a) aims at minimizing the number of consumers not served by any route; therefore, the number of consumers served y^* is given by $|\mathcal{C}| - \delta^*$. Constraints (2b) ensure that each service point is visited at most once by the selected routes. Constraint (2c) guarantees that at most $|\mathcal{K}|$ routes are selected. Constraints (2d) model that each class of consumers h is either visited (in this case, $\sum_{r \in \mathcal{R}} (\sum_{i \in \mathcal{S}(\mathcal{H}_h)} a_{ir}) \xi_r \geq 1$ and $\delta_h = 0$) or not (in this case, $\sum_{r \in \mathcal{R}} (\sum_{i \in \mathcal{S}(\mathcal{H}_h)} a_{ir}) \xi_r = 0$ and $\delta_h = 1$). Constraints (2e) and (2f) define the range of the decision variables.

Let us call $\overline{\mathrm{LF}}$ the linear relaxation of $\overline{\mathrm{F}}$, and let $z(\overline{\mathrm{LF}})$ be its optimal solution cost.

4.1.2. Pricing Problem. Let $u_i \in \mathbb{R}_-$ be the dual variable associated with constraint (2b) of service point $i \in \mathcal{S}$ of $\overline{\mathrm{LF}}$, let $u_0 \in \mathbb{R}_-$ be the dual variable associated with constraint (2c), and let $v_h \in \mathbb{R}_+$ be the dual variable of class $h = 1, \ldots, |\mathbb{H}|$ of constraint (2d). The dual of $\overline{\mathrm{LF}}$ is

$$\begin{aligned} z(\mathrm{D}) = \max \; & \sum_{i \in \mathcal{S}} u_i + |\mathcal{K}| u_0 + \sum_{h=1}^{|\mathbb{H}|} v_h \\ \text{s.t.} \; & \sum_{i \in \mathcal{S}} a_{ir} u_i + u_0 \\ & \quad + \sum_{h=1}^{|\mathbb{H}|} \left(\sum_{i \in \mathcal{S}(\mathcal{H}_h)} a_{ir} \right) v_h \leq 0 \qquad r \in \mathcal{R}, \\ & 0 \leq v_h \leq |\mathcal{H}_h| \qquad h = 1, \ldots, |\mathbb{H}|, \\ & u_i \in \mathbb{R}_- \qquad i \in \mathcal{V}. \end{aligned}$$

Given the dual variables $(\boldsymbol{u}, \boldsymbol{v})$, the pricing problem corresponds to finding the minimum reduced cost route, where the reduced cost $c_r(\boldsymbol{u}, \boldsymbol{v})$ of route $r \in \mathcal{R}$ is defined as $c_r(\boldsymbol{u}, \boldsymbol{v}) = -\sum_{i \in \mathcal{S}} a_{ir} u_i - u_0 - \sum_{h=1}^{|\mathbb{H}|} (\sum_{i \in \mathcal{S}(\mathcal{H}_h)} a_{ir}) v_h$.

Solving the pricing problem corresponds to finding a route of minimum reduced cost, that is, solving the problem $\min_{r \in \mathcal{R}} c_r(\boldsymbol{u}, \boldsymbol{v})$. This problem corresponds to the well-known elementary shortest path problem with resource constraints, which is known to be NP-hard in the strong sense (Dror 1994). To simplify the pricing problem, it is common in the literature to price out *ng*-routes, thus allowing routes to visit some of the service points more than once and allow some subtours. The *ng*-path relaxation was introduced in Baldacci, Mingozzi, and Roberti (2011) and applied to the TOPO as follows.

Let $\mathcal{N}_i \subseteq \mathcal{S}$ be an a priori–defined set of selected service points associated with service point $i \in \mathcal{N}$ such that $i \in \mathcal{N}_i$. With each path $P = (0, i_1, i_2, \ldots, i_{\ell(P)})$ that starts from the depot, visits a set of $\ell(P)$ service points, and ends at service point $i_{\ell(P)}$, we associate a set $NG(P) \subseteq \mathcal{N}_{i_{\ell(P)}}$ defined as follows:

$$NG(P) = \left\{ i_k, k = 1, \ldots, \ell(P) - 1 \,|\, i_k \in \bigcap_{j=k+1}^{\ell(P)} \mathcal{N}_{i_j} \right\} \cup \{i_{\ell(P)}\}; \tag{4}$$

that is, the set $NG(P)$ contains the final vertex $i_{\ell(P)}$ visited by path P and any other vertex i_k, $k = 1, \ldots, \ell(P) - 1$, that belongs to all sets $\mathcal{N}_{i_{k+1}}, \mathcal{N}_{i_{k+2}}, \ldots, \mathcal{N}_{i_{\ell(P)}}$ associated with the vertices visited by path P after visiting i_k. Note that

without forcing i to be included in $\mathcal{N}_i$, it would not have been possible to define $NG(P)$ as a subset of $\mathcal{N}_{i_{\ell(P)}}$.

An *ng*-path (NG, t, i) is a nonnecessarily elementary path $P = (0, i_1, i_2, \ldots, i_{\ell(P)})$ that starts from the depot, visits a set of service points such that $NG(P) = NG$, has a total duration of t, and ends at vertex $i \in \mathcal{N}$ such that $i \notin NG(P')$, where $P' = (0, i_1, i_2, \ldots, i_{\ell(P)-1})$ is an *ng*-path. Any $(NG, t, 0)$-path ending at the depot at time $t > 0$ is called an *ng*-route.

Let $f(NG, t, i)$ be the cost of a least cost *ng*-path (NG, t, i). Functions $f(NG, t, i)$ can be computed by using dynamic programming as follows. Define the state set $\mathbb{S} = \{(NG, t, i) : i \in \mathcal{V},\ t \in [0, T],\ NG \subseteq \mathcal{N}_i \text{ s.t. } i \in NG\}$. For each state $(NG, t, i) \in \mathbb{S}$, function $f(NG, t, i)$ is computed as

$$f(NG, t, i) = \min_{(NG', t', j) \in \Gamma(NG, t, i)} \{f(NG', t', j) + c_{ji}(\boldsymbol{u}, \boldsymbol{v})\}, \quad (5)$$

where the set $\Gamma(NG, t, i)$ contains the subset of predecessor states of (NG, t, i) defined as $\Gamma(NG, t, i) = \{(NG', t', j) \in \mathbb{S} \mid t' = t - t_{ji},\ j \in \mathcal{V} \setminus \{i\}, NG' \subseteq \mathcal{N}_j : j \in NG' \text{ and } NG' \cap \mathcal{N}_i = NG \setminus \{i\}\}$, and $c_{ji}(\boldsymbol{u}, \boldsymbol{v})$ is the reduced cost of arc $(j, i) \in \mathcal{A}$ with respect to (w.r.t.) the dual solution $(\boldsymbol{u}, \boldsymbol{v})$ defined as

$$c_{ji}(\boldsymbol{u}, \boldsymbol{v}) = -u_i - \sum_{h \in \mathcal{H}(i)} v_h. \quad (6)$$

To compute functions $f(NG, t, i)$, the following initialization is required: $f(\emptyset, 0, 0) = 0$ and $f(\emptyset, t, 0) = \infty$, $t \in (0, T]$.

The number of functions $f(NG, t, i)$ to compute can be reduced by applying the following dominance rule.

Dominance Rule 1. *Let* $(NG, t, i), (NG', t', i) \in \mathbb{S}$ *be two states such that* (1) $f(NG, t, i) \le f(NG', t', i)$, (2) $t \le t'$, *and* (3) $NG \subseteq NG'$ *(and such that one of the three conditions is strictly satisfied). Then state* (NG, t, i) *dominates state* (NG', t', i).

4.1.3. Speeding Up the Pricing Problem Solution.

We use three acceleration techniques to speed up the pricing problem solution algorithm:

1. *Bidirectional ng-path* (Righini and Salani 2006). *ng*-routes are priced out by generating *ng*-paths up to a halfway point, and paths are combined to generate routes. We can use the time to set the halfway point and stop propagating a path as soon as the arrival time $t(P)$ at the last customer i of an *ng*-path (NG, t, i) exceeds $\lceil \frac{T}{2} \rceil$. A forward *ng*-path (NG_1, t_1, i_1) can be combined with a backward *ng*-path (NG_2, t_2, i_2) if either (NG_1, t_1, i_1) ends at the depot or $t_1 > \lceil \frac{T}{2} \rceil$, $i_1 = i_2$, $NG_1 \cap NG_2 = \{i_1\}$, and $t_1 + t_2 \le T$. Moreover, the reduced cost $c_r(\boldsymbol{u}, \boldsymbol{v})$ of the route obtained by combining paths P_1 and P_2 $(r = P_1 \oplus P_2)$ is $c_r(\boldsymbol{u}, \boldsymbol{v}) = f(NG_1, t_1, i_1) + f(NG_2, t_2, i_2)$.

2. *Heuristic pricing*. Before solving the pricing problem to optimality, two heuristics are applied in sequence to eventually generate negative reduced cost columns. Both the heuristics apply Dominance Rule 1 by ignoring the condition $NG_1 \subseteq NG_2$, relaxing then the criteria for the dominance. In addition, the first heuristic propagates each state (NG, t, i) toward the service points corresponding to the η_A arcs outgoing from i and having the lowest reduced costs $c_{ij}(\boldsymbol{u}, \boldsymbol{v})$. The second heuristic is run only if the first does not succeed in finding negative reduced cost columns. When both heuristics fail, the pricing problem is solved to optimality.

3. *Dynamic ng-path* (Roberti and Mingozzi 2014). The idea is to compute increasingly better lower bounds by starting from small sets $\mathcal{N}_i$ and iteratively adding service points to the sets $\mathcal{N}_i$ based on the optimal fractional solution computed for $\overline{\text{LF}}$ with the current sets $\mathcal{N}_i$. In particular, the initial sets $\mathcal{N}_i$ contain the $\underline{\eta}_{N_i}$ service points closest to i, and at each iteration, the sets $\mathcal{N}_i$ are updated to eliminate the first shortest cycle contained in each route of the optimal fractional solution. The process iterates until either all the routes of the optimal fractional solution are cycle-free or all the possible updates would increase the cardinality of a set $\mathcal{N}_i$ beyond a given maximum value $\overline{\eta}_{N_i}$.

4.1.4. Valid Inequalities.

The linear relaxation of (2a)–(2f) can be strengthened by adding the well-known SR inequalities introduced by Jepsen et al. (2008) for the vehicle routing problem with time windows. As commonly done, we consider SRs defined over triples of service points only, which is

$$\sum_{r \in \mathcal{R}} \left\lfloor \frac{a_{ir} + a_{jr} + a_{kr}}{2} \right\rfloor \xi_r \le 1 \qquad \{i, j, k\} \subseteq \mathcal{S} \quad (7)$$

that can easily be separated by pure enumeration. To handle these nonrobust cuts, the pricing problem solution algorithm is modified as illustrated in Jepsen et al. (2008).

4.1.5. Restricted Master Heuristic.

To speed up the branch-and-price algorithm, we embed a restricted master heuristic (Joncour et al. 2010) in the solution framework. The basic idea behind restricted master heuristics is to solve, by means of a general mixed integer linear programming (MILP) solver, the master problem ($\overline{\text{F}}$) restricted to a subset of the generated columns. By removing constraints (2b) from model $\overline{\text{F}}$, any subset of generated columns (even including columns corresponding to routes with cycles) can be used to compute an infeasible solution (optimal w.r.t. the considered columns) that can be easily made feasible (without worsening its value) by removing all but one visit for each service point visited multiple times. Thus, each time we compute an optimal fractional

solution for $\overline{\text{LF}}$, the columns defining the solution are used as they are to initialize the restricted master problem (from which constraints (2b) are removed), which is then solved by means of a general MILP solver. The heuristic is run up to a given level of the branch-and-bound tree as specified in Section 6.

4.1.6. Branching. We perform a traditional binary branching according to two hierarchical rules. The first rule is on whether a service point is visited or not. The second rule is on the arcs to traverse.

4.2. Improved Branch-and-Cut-and-Price Algorithms

The integrality gap of route-based model $\overline{\text{F}}$ presented in Section 4.1.1 is strongly affected by the coefficients of variables ξ_r in constraints (2d). Indeed, in constraint (2d) of a given class h ($h = 1, \ldots, |\mathbb{H}|$), the variable ξ_r associated with route $r \in \mathcal{R}$ has a coefficient $\overline{a}_{hr}$ equal to the number of times the service points that can serve the class h (i.e., $\overline{a}_{hr} = \sum_{i \in \mathcal{S}(\mathcal{H}_h)} a_{ir}$) are visited in the route.

Let us associate with each route r and each class h a binary coefficient $\underline{a}_{hr}$ defined as

$$\underline{a}_{hr} = \begin{cases} 1 & \text{if } \overline{a}_{hr} \geq 1, \\ 0 & \text{otherwise;} \end{cases}$$

that is, $\underline{a}_{hr}$ is equal to 1 if at least one of the service points that can serve the class h is visited by route r (0 otherwise). An alternative formulation for the TOPO is obtained from $\overline{\text{F}}$ by replacing in constraints (2d) the coefficients $\overline{a}_{hr} = \sum_{i \in \mathcal{S}(\mathcal{H}_h)} a_{ir}$ with $\underline{a}_{hr}$. The resulting formulation $\underline{\text{F}}$ is

$$[\underline{\text{F}}] \quad \delta^* = \min \sum_{h=1}^{|\mathbb{H}|} |\mathcal{H}_h| \delta_h \tag{8a}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} a_{ir} \xi_r \leq 1 \qquad i \in \mathcal{S}, \tag{8b}$$

$$\sum_{r \in \mathcal{R}} \xi_r \leq |\mathcal{K}|, \tag{8c}$$

$$\sum_{r \in \mathcal{R}} \underline{a}_{hr} \xi_r + \delta_h \geq 1 \qquad h = 1, \ldots, |\mathbb{H}|, \tag{8d}$$

$$\xi_r \in \{0, 1\} \qquad r \in \mathcal{R}, \tag{8e}$$

$$\delta_h \in \mathbb{R}_+ \qquad h = 1, \ldots, |\mathbb{H}|. \tag{8f}$$

It is easy to observe that the linear relaxation of $\underline{\text{F}}$ (hereafter $\underline{\text{LF}}$) provides a lower bound $z(\underline{\text{LF}})$ on the TOPO that is greater than or equal to $z(\overline{\text{LF}})$. Nevertheless, it is computationally challenging to solve the pricing problem of $\underline{\text{F}}$ because the definition of coefficients $\underline{a}_{hr}$ requires keeping track of all service points visited by a route to properly take into account the dual variables $\boldsymbol{v}$. Therefore, even computing lower bounds to $z(\underline{\text{LF}})$ by pricing out nonelementary routes (such as *ng*-routes) may be computationally prohibitive.

In Sections 4.2.1 and 4.2.2, we propose two alternative ways of defining coefficients to replace $\overline{a}_{hr}$ in constraints (2d) that are computationally tractable when pricing out *ng*-routes and provide lower bounds that can be worse than $z(\underline{\text{LF}})$ but can be (significantly) better than $z(\overline{\text{LF}})$. Each of the definitions leads to an improved route-based model that can be used in place of model $\overline{\text{F}}$, and to an alternative, potentially improving version of the BCP algorithm discussed in Section 4.1.1.

4.2.1. First Improved BCP Algorithm. In the first computationally tractable improved route-based model (hereafter called F′), for each route $r \in \mathcal{R}$, we count the number of times a class h can be served by a service point j (i.e., $j \in \mathcal{S}(\mathcal{H}_h)$) visited in the route and the service point i visited right before j in the route cannot serve class h (i.e., $i \notin \mathcal{S}(\mathcal{H}_h)$). Therefore, given route $R = (0, i_1, i_2, \ldots, i_{\ell(R)}, 0)$, the number of times a'_{hr} class $h = 1, \ldots, |\mathbb{H}|$ is served by route R is computed as $a'_{hr} = |\{i_k, k = 1, \ldots, \ell(R) \,|\, i_k \in \mathcal{S}(\mathcal{H}_h),\ i_{k-1} \notin \mathcal{S}(\mathcal{H}_h)\}|$.

We can observe that $\underline{a}_{hr} \leq a'_{hr} \leq \overline{a}_{hr}$ for each class $h = 1, \ldots, |\mathbb{H}|$ and each route $r \in \mathcal{R}$. Let us call F′ the formulation obtained from $\underline{\text{F}}$ by replacing coefficients $\underline{a}_{hr}$ with coefficients a'_{hr} in constraints (8d). Moreover, let LF′ be the linear relaxation of F′, and let $z(\text{LF}')$ be its optimal solution cost. We can observe that $z(\overline{\text{LF}}) \leq z(\text{LF}') \leq z(\underline{\text{LF}})$.

The pricing problem to compute $z(\text{LF}')$ can be solved by using the same methods used to compute $z(\overline{\text{LF}})$ by redefining the arc reduced costs (6) as

$$c'_{ji}(\boldsymbol{u}, \boldsymbol{v}) = -u_i - \sum_{h \in \mathcal{H}(i) \setminus \mathcal{H}(j)} v_h. \tag{9}$$

4.2.2. Second Improved BCP Algorithm. In the second computationally tractable improved route-based model (hereafter called F″), the number of times a route $r \in \mathcal{R}$ serves class $h = 1, \ldots, |\mathbb{H}|$ depends on the sets $\mathcal{N}_i \subseteq \mathcal{S}$ of selected service points associated with each service point $i \in \mathcal{S}$. As in definition (4), for each service point i_q visited along a route $r = (0, i_1, i_2, \ldots, i_{\ell(r)}, 0)$, let us define a set of service points $NG_q(r)$ associated with i_q as $NG_q(r) = \{i_k, k = 1, \ldots, q-1 \,|\, i_k \in \cap_{j=k+1}^{q} \mathcal{N}_{i_j},\} \cup \{i_q\}$. The number of times a class h is served by route r is then counted as the number of visits to a service point i_q that can serve class h (i.e., $i_q \in \mathcal{S}(\mathcal{H}_h)$) and such that none of the service points in the set $NG_{q-1}(r)$ can serve class h (i.e., $NG_{q-1}(r) \cap \mathcal{S}(\mathcal{H}_h) = \emptyset$), which is $a''_{hr} = \left|\{i_q, q = 1, \ldots, \ell(r) \,|\, i_q \in \mathcal{S}(\mathcal{H}_h), NG_{q-1}(r) \cap \mathcal{S}(\mathcal{H}_h) = \emptyset\}\right|$. We can observe that $\underline{a}_{hr} \leq a''_{hr} \leq a'_{hr} \leq \overline{a}_{hr}$. In particular, $\underline{a}_{hr} = a''_{hr}$ for each $h = 1, \ldots, |\mathbb{H}|$ and for each $r \in \mathcal{R}$ if $\mathcal{N}_i = \mathcal{S}$ for each $i \in \mathcal{S}$, and $a''_{hr} = a'_{hr}$ for each $h = 1, \ldots, |\mathbb{H}|$ and for each $r \in \mathcal{R}$ if $\mathcal{N}_i = \{i\}$ for each $i \in \mathcal{S}$.

Let us call F″ the formulation obtained from $\underline{\text{F}}$ by replacing coefficients $\underline{a}_{hr}$ with coefficients a''_{hr} in constraints (8d). Moreover, let LF″ be the linear relaxation of F″, and let $z(\text{LF}'')$ be its optimal solution cost. We can observe that $z(\text{LF}') \leq z(\text{LF}'') \leq z(\underline{\text{LF}})$.

Moreover, $z(\mathrm{LF}') = z(\mathrm{LF}'')$ if $\mathcal{N}_i = \{i\}$ for each $i \in \mathcal{S}$, and $z(\mathrm{LF}'') = z(\underline{\mathrm{LF}})$ if $\mathcal{N}_i = \mathcal{S}$ for each $i \in \mathcal{S}$.

The pricing problem to compute $z(\mathrm{LF}'')$ can be solved by using the same methods used to compute $z(\overline{\mathrm{LF}})$ by redefining recursion (5) as follows: $f(NG, t, i) = \min_{(NG', t', j) \in \Gamma(NG, t, i)} \{f(NG', t', j) + c''_{ji}(NG', \boldsymbol{u}, \boldsymbol{v})\}$, where $c''_{ji}(NG', \boldsymbol{u}, \boldsymbol{v})$ is the cost of a propagation along arc $(j, i) \in \mathcal{A}$ when $NG_j(r) = NG'$, defined as

$$c''_{ji}(NG', \boldsymbol{u}, \boldsymbol{v}) = -u_i - \sum_{h \in \mathcal{H}(i) : h \notin \bigcup_{k \in NG'} \mathcal{H}(k)} v_h. \tag{10}$$

The use of sets $NG_q(r)$ to compute coefficients a''_{hr} implies a monodirectional propagation of the states. Actually, by propagating states backward along a route, it is possible to define column coefficients that are different from those computed by using forward propagation. Consider the example of a route $r = (0, i_1, i_2, i_3, 0)$, with $i_1 \in \mathcal{N}_{i_2}$, $i_3 \notin \mathcal{N}_{i_2}$, $\mathcal{H}(i_1) \cap \mathcal{H}(i_2) = \mathcal{H}(i_2) \cap \mathcal{H}(i_3) = \emptyset$, and $\mathcal{H}(i_1) \cap \mathcal{H}(i_3) = \{\bar{h}\}$. The forward propagation of the states along the route implies $a''_{\bar{h}r} = 1$, whereas $a''_{\bar{h}r} = 2$ when states are propagated backward. Given this asymmetry in the solution spaces implied by the forward and backward propagation of the states, the bidirectional acceleration technique is not applied while solving the pricing problem arising for F″.

To our knowledge, this is the first time that *ng*-paths are used not just to eliminate cycles but also to improve the linear relaxation bound by lifting some of the column coefficients. This benefit comes at the expense of an increase in the computational time needed to manage the propagation of the states and the definition of the column coefficients. Such an idea may be successfully applied to other routing problems where vehicles can fulfill (multiple) tasks by visiting locations and each task can be fulfilled by visiting a subset of all locations; this happens, for example, in applications of the generalized rural postman problem and its generalizations (see Drexl 2007).

5. A Large Neighborhood Search Metaheuristic

In this section, we present a large neighborhood search metaheuristic for the TOPO. Since its introduction by Shaw (1998), LNS has proved to be an efficient tool for solving many vehicle routing problems inspired by real-life applications (e.g., Ropke and Pisinger 2006; Adulyasak, Cordeau, and Jans 2012; Masson, Lehuédé, and Péton 2013; Emde and Schneider 2018; Hübner and Ostermeier 2018). According to Pisinger and Ropke (2019), the main idea of LNS is to start from an initial solution and gradually improve it by alternately applying a destroy method and a repair method to the incumbent solution. The destroy method significantly changes the incumbent solution to guarantee diversification. The repair method intensifies the search in the neighborhood of the incumbent solution. An exhaustive review of LNS and related metaheuristics is provided by Pisinger and Ropke (2019).

5.1. Overview of the Proposed LNS

LNS (Algorithm 1) starts with generating an initial solution $\mathcal{X}$ (Line 1) by using an adaptation of the well-known nearest-neighbor approach of Solomon (1987). In particular, a route is created for each vehicle by iteratively inserting the unvisited service point that minimizes the extra mileage until no more service points can be added. The initial solution obtained is then added to the pool of best solutions found, Ω, and also represents the best-known solution, $\mathcal{X}^*$ (Line 2).

The core of LNS is represented by the main loop (Lines 3–11), which is iterated $\overline{\eta}_1$ times, where $\overline{\eta}_1$ is a parameter. At each iteration, the first step is to randomly choose a solution $\mathcal{X}$ from the pool (Line 4) according to a uniform probability distribution. The second step is to destroy solution $\mathcal{X}$ (Line 5) by randomly removing a percentage of the visited service points. This percentage is chosen according to a uniform distribution within two intervals $[\alpha_1, \beta_1]$ and $[\alpha_2, \beta_2]$, where $\alpha_1 < \beta_1 \ll \alpha_2 < \beta_2$. A percentage in the interval $[\alpha_1, \beta_1]$ intensifies the search around the incumbent solution, whereas a percentage in the interval $[\alpha_2, \beta_2]$ diversifies the search. The service points to be removed are randomly selected with an equal probability. Then, the solution returned by the destroy method is improved by applying a repair method consisting of a local search procedure (Line 6), a random replacement of some of the visited service points (Line 7), and again the local search procedure (Line 8). More details on these procedures are provided in the next sections and in the pseudocode descriptions of Algorithms 2 and 3. At the end of each iteration, the best solution found, $\mathcal{X}^*$, is updated (Line 9) along with the pool of solutions, Ω (Line 10), to ensure that it contains the ω best solutions found by LNS, ω being a parameter.

Algorithm 1 (Overview of LNS)
Input: TOPO input data
Parameters: max iterations, $\overline{\eta}_1$
Output: TOPO solution $\mathcal{X}^*$
1. $\mathcal{X} \leftarrow$ *generateInitialSolution*
2. $\mathcal{X}^* \leftarrow \mathcal{X}$, $\Omega \leftarrow \{\mathcal{X}\}$
3. **for** $\eta_1 = 1, \ldots, \overline{\eta}_1$ **do**
4. $\quad \mathcal{X} \leftarrow$ *randomlySelectFromPool*(Ω)
5. $\quad \mathcal{X} \leftarrow$ *destroySolution*$(\mathcal{X})$
6. $\quad \mathcal{X} \leftarrow$ *applyLocalSearch*$(\mathcal{X})$ ▷ See Algorithm 2
7. $\quad \mathcal{X} \leftarrow$ *replaceRandom*$(\mathcal{X})$ ▷ See Algorithm 3
8. $\quad \mathcal{X} \leftarrow$ *applyLocalSearch*$(\mathcal{X})$
9. $\quad \mathcal{X}^* \leftarrow$ *updateBest*$(\mathcal{X}^*, \mathcal{X})$
10. $\quad \Omega \leftarrow$ *updatePool*$(\Omega, \mathcal{X})$
11. **end for**
12. **return** $\mathcal{X}^*$

5.2. Local Search

The local search procedure applied at each iteration of LNS is detailed in Algorithm 2. The input is a solution $\mathcal{X}$, and the output is the solution obtained by iteratively applying a sequence of local search operators. The local search procedure is repeated as long as the incumbent solution is improved by any of the operators.

Algorithm 2 (*applyLocalSearch*($\mathcal{X}$))
Input: solution $\mathcal{X} = (R_1, R_2, \ldots, R_{|\mathcal{K}|})$
Output: new solution $\mathcal{X}$ improved with local search

do
  $\mathcal{X}' \leftarrow \mathcal{X}$
  do ▷ Routing improvement
    $\mathcal{X}'' \leftarrow \mathcal{X}$
    $\mathcal{X} \leftarrow$ *2-opt-intra*($\mathcal{X}$)
    $\mathcal{X} \leftarrow$ *OR-opt2-intra*($\mathcal{X}$)
    $\mathcal{X} \leftarrow$ *swap-inter*($\mathcal{X}$)
    $\mathcal{X} \leftarrow$ *relocate-inter*($\mathcal{X}$)
    $\mathcal{X} \leftarrow$ *2-1-exchange-inter*($\mathcal{X}$)
  while $\mathcal{X} \neq \mathcal{X}''$
  do ▷ Profit improvement
    $\mathcal{X}'' \leftarrow \mathcal{X}$
    $\mathcal{X} \leftarrow$ *replace*($\mathcal{X}$)
    $\mathcal{X} \leftarrow$ *add*($\mathcal{X}$)
  while $\mathcal{X} \neq \mathcal{X}''$
while $\mathcal{X} \neq \mathcal{X}'$
return $\mathcal{X}$

The local search procedure consists of two main phases (i.e., a *routing improvement* phase and a *profit improvement* phase), each one iterated until a local minimum is reached. The routing improvement phase does not change the set of visited service points (and therefore does not change the profit of the solution) and aims at minimizing the length of the shortest route in the incumbent solution. To this end, five local search operators are applied: (1) *2-opt-intra* (i.e., the well-known 2-opt operator applied within a route), (2) *OR-opt2-intra* (i.e., shifting sequences of two consecutive service points forward and backward in the corresponding route), (3) *swap-inter* (i.e., exchanging two service points of two different routes), (4) *relocate-inter* (i.e., moving a service point from its route to any position in any other route), and (5) *2-1-exchange-inter* (i.e., exchanging a service point of a route with two service points of another route). The profit improvement phase aims at increasing the total profit of the visited service points by applying two operators that change the set of service points: (1) *replace*, which replaces a visited service point with an unvisited service point, and (2) *add*, which inserts unvisited service points in any of the routes.

Notice that computing the profit for a solution of the two profit-improving operators can be time consuming. Let $f_{\text{profit}}(S)$ be the total profit of the service points of the set $S \subseteq \mathcal{S}$, and let us call S the set of service points visited by the incumbent solution $\mathcal{X}$. A move of the *replace* operator tries to replace a service point $i \in S$ with a service point $j \in \mathcal{S} \setminus S$. Unfortunately, we can observe that the profit of the set of service points $(S \setminus \{i\}) \cup \{j\}$ may not be $f_{\text{profit}}(S) - f_{\text{profit}}(\{i\}) + f_{\text{profit}}(\{j\})$, indeed $f_{\text{profit}}((S \setminus \{i\}) \cup \{j\}) \leq f_{\text{profit}}(S) - f_{\text{profit}}(\{i\}) + f_{\text{profit}}(\{j\})$. Similarly, a generic move of the *add* operator tries to add a service point $j \in \mathcal{S} \setminus S$ to the routes of $\mathcal{X}$, but we can observe that $f_{\text{profit}}(S \cup \{j\})$ may be less than $f_{\text{profit}}(S) + f_{\text{profit}}(\{j\})$ (i.e., $f_{\text{profit}}(S \cup \{j\}) \leq f_{\text{profit}}(S) + f_{\text{profit}}(\{j\})$).

Therefore, to limit the computation time for evaluating the *replace* and *add* operators, we apply a memory-intensive preprocessing step that precomputes the effect of adding/removing a service point to/from a set of visited service points on the total profit. In particular, for each service point $i \in \mathcal{S}$, let $\mathbb{O}(i) \subseteq \mathcal{S} \setminus \{i\}$ be the set of service points that serve at least one of the consumers served by service point i, that is, $\mathbb{O}(i) = \{j \in \mathcal{S} \setminus \{i\} \mid \mathcal{C}_i \cap \mathcal{C}_j \neq \emptyset\}$. Let $\mathcal{P}(\mathbb{O}(i))$ be the power set of $\mathbb{O}(i)$. For each subset of service points $O \in \mathcal{P}(\mathbb{O}(i))$, we precompute the difference $\Delta(O, i)$ between the profit of the set of service points $O \cup \{i\}$ and O, that is, $\Delta(O, i) = f_{\text{profit}}(O \cup \{i\}) - f_{\text{profit}}(O)$. If it is possible to compute the values $\Delta(O, i)$ for each $i \in \mathcal{S}$ and each $O \in \mathcal{P}(\mathbb{O}(i))$, then the profit of any *replace* and *add* move during LNS can easily be computed in constant time: any *replace* move that replaces the visited service point $i \in S$ with the unvisited service points $j \in \mathcal{S} \setminus S$ achieves a solution with profit equal to $f_{\text{profit}}(S) - \Delta(S \cap \mathbb{O}(i), i) + \Delta((S \setminus \{i\}) \cap \mathbb{O}(j), j)$, and any *add* move that adds unvisited service point $j \in \mathcal{S} \setminus S$ to the visited service points S achieves a solution with profit equal to $f_{\text{profit}}(S) + \Delta(S \cap \mathbb{O}(j), j)$. Clearly, if the cardinality of the sets $\mathbb{O}(i)$ is high, precomputing all values $\Delta(O, i)$ may become prohibitive. In such a case, other strategies to estimate the increase (decrease) in the profit of the incumbent solution derived from adding (removing) a service point need to be devised. For example, one could create a subset of the service points $\hat{\mathbb{O}}(i) \subset \mathbb{O}(i)$ such that it is possible to generate the power set $\mathcal{P}(\hat{\mathbb{O}}(i))$. In our computational experiments, we do not investigate the case where all the values $\Delta(O, i)$ cannot be computed a priori because of memory limitations because such an issue does not appear in the real-life cases we consider.

5.3. Random Replacement

The *replaceRandom* procedure is performed once at each iteration of LNS to increase the profit of the incumbent solution. The input of the procedure is a solution $\mathcal{X} = (R_1, R_2, \ldots, R_{|\mathcal{K}|})$, and the output is another solution with possibly a higher profit. The procedure has two parameters, $\overline{\eta}_2$ and $\overline{\eta}_3$, which control the number of times a route can be changed to improve

its profit and the maximum number of changes that can be performed every time, respectively. A detailed pseudocode of the *replaceRandom* procedure is provided in Algorithm 3.

Algorithm 3 (*replaceRandom*($\mathcal{X}$))
Input: solution $\mathcal{X} = (R_1, R_2, \ldots, R_{|\mathcal{R}|})$
Parameters: max iterations of changes per route, $\overline{\eta}_2$; max changes per iteration, $\overline{\eta}_3$
Output: new solution $\mathcal{X}$
1. **for** $r = 1, \ldots, |\mathcal{R}|$ **do**
2. **for** $\eta_2 = 1, \ldots, \overline{\eta}_2$ **do**
3. Randomly select a set $\tilde{\mathcal{S}}$ of η_3 service points from route R_r, $\eta_3 \sim U[1, \min\{\overline{\eta}_3, \ell(R_r)\}]$
4. **if** $f_{\text{profit}}(\tilde{\mathcal{S}}) < f_{\text{profit}}(\mathcal{S} \setminus S)$ **then**
5. Remove the service points $\tilde{\mathcal{S}}$ from R_r
6. Randomly insert as many unvisited service points as possible into R_r
7. **if** the removals and insertions of service points decreased the profit of R_r **then**
8. Undo the changes of R_r
9. **end if**
10. **end if**
11. **end for**
12. **end for**
13. **return** $\mathcal{X}$

The *replaceRandom* procedure focuses on one route at a time (Line 1). For each route r, $\overline{\eta}_2$ attempts to improve the total profit are made (Line 2). In each attempt, a set $\tilde{\mathcal{S}}$ of η_3 service points visited by route R_r is randomly selected with a uniform distribution (Line 3); η_3 is a random number between 1 and $\min\{\overline{\eta}_3, \ell(R_r)\}$, where $\overline{\eta}_3$ is a parameter and $\ell(R_r)$ represents the number of service points visited by route R_r. The service points $\tilde{\mathcal{S}}$ are candidates to leave route R_r. If the total profit of the service points $\tilde{\mathcal{S}}$ is smaller than the profit of the unvisited $(\mathcal{S} \setminus S)$ service points (Line 4), then, first, the service points $\tilde{\mathcal{S}}$ are removed from route R_r and marked as unvisited (Line 5), and second, the unvisited service points are iteratively inserted into route R_r (in a random order) in the position that minimizes the distance to visit them (Line 6). If the removal and insertion operations on route R_r increase its profit (Line 6), then the changes are maintained; otherwise, the changed are undone (Line 7).

6. Computational Results

This section presents a computational analysis of the branch-and-cut-and-price algorithms described in Section 4 and of the LNS metaheuristic presented in Section 5. For the sake of brevity, only aggregated results are presented and discussed in this section. Detailed results can be found in the e-companion of this paper.

All proposed algorithms have been implemented in C++; compiled with Visual Studio 2017, 64 bit; and tested on a single core of an Intel Core i7-6700U running at 4.00 GHz, equipped with 24 GB of memory. Moreover, for the BCP algorithms, at each column generation iteration, the linear relaxation of model $\overline{\text{F}}$, F′, or F″ is solved with CPLEX 12.7, which is also used as a general MILP solver in the restricted master heuristic (see Section 4.1.5).

All the BCP algorithms share the following setting. The first heuristic to solve the pricing problem limits the number of propagations of a state to $\eta_A = 5$ service points. The dynamic *ng*-path technique is applied up to the third level of the branch-and-bound tree, with $\underline{\eta}_{N_i} = 3$ and $\overline{\eta}_{N_i} = 11$. Once the dynamic update of the sets $\mathcal{N}_i$ fails while processing a node, sets $\mathcal{N}_i$ cannot be further updated for that node. At most one SR inequality (7) at a time can be added to the considered model (i.e., the inequality associated with the greatest violation value, with a minimum violation threshold of 0.01), and at most two inequalities per branch-and-bound node. The separation algorithm checking for violated SR inequalities is run up to the tenth level of the branch-and-bound tree, for a maximum of 20 SR inequalities per subtree. In particular, the separation algorithm can be run only if the dynamic update of the sets $\mathcal{N}_i$ fails (or when the update of the sets is no more allowed). The restricted master heuristic is applied up to the tenth level of the branch-and-bound tree. Finally, branching is done on the variable associated with the most fractional value.

For the LNS, unless stated otherwise, the following parameter settings are used: $[\alpha_1, \beta_1] = [20\%, 30\%]$, $[\alpha_2, \beta_2] = [80\%, 90\%]$, $\omega = 50$, $\overline{\eta}_1 = 10000 - \lfloor \frac{\max\{0, 100 - |\mathcal{S}|\}}{10} \rfloor \cdot 500$, $\overline{\eta}_2 = 30$, and $\overline{\eta}_3 = 3$.

Hereafter, we will refer to the BCP algorithms based on models $\overline{\text{F}}$, F′, and F″ as `Baseline`, `BCP1`, and `BCP2`, respectively. Computational times are reported in seconds throughout this section. Furthermore, to simplify the comparison with the literature, all primal and dual bounds are reported as if the objective function of the TOPO is to maximize the number of consumers served (as defined in (1a)) instead of minimize the number of consumers that are not served (as defined in (2a) and (8a)). Indeed, as observed in Section 4, objective functions (2a) and (8a) can also be equivalently formulated as $\min\{|\mathcal{C}| - \delta^*\}$.

The computational analysis is conducted on three sets of test instances. The first set (discussed in Section 6.1) consists of 215 synthetic TOPO instances derived from the instances by Chao, Golden, and Wasil (1996) commonly used to assess the performance of solution methods for the TOP. The second set (discussed in Section 6.2) consists of the 387 Chao TOP instances, which, as explained in Section 3, is a special case of

the TOPO. The third set (discussed in Section 6.3) consists of 10 real-life instances of the TOPO that are faced by Geldmaat in their cash supply chain, which motivated the research on the TOPO. For all instances, travel times are rounded to the nearest value with a precision of 1.0e-12, and postprocessing is applied to enforce the triangular inequality. The first two sets of instances are available upon request. For confidentiality reasons, we cannot disclose the third set of instances.

6.1. Computational Results on Synthetic TOPO Instances

6.1.1. Description of the Instances. The first set of instances consists of 215 synthetic TOPO instances we derived from the well-known TOP instances introduced by Chao, Golden, and Wasil (1996). The Chao instances include seven families of instances (numbered from 1 to 7), each with a fixed number of service points, overall ranging from 19 to 100. Each family of instances consists of three groups of instances, where each group contains the same number of instances and is characterized by a different number of vehicles (two, three, or four). Each instance of the resulting 21 groups is further characterized by a different maximum route duration T. It should be noted that in instances with small values of T, some of the service points may not be reachable. Table 1 summarizes the main features of each family, namely, the number of service points ($|\mathcal{S}|$), number of instances (nInst), and minimum and maximum route durations of the instances in each group ($T_{|\mathcal{K}|=2}$, $T_{|\mathcal{K}|=3}$, and $T_{|\mathcal{K}|=4}$).

To generate the 215 synthetic TOPO instances, we selected for each group of the Chao instances up to three TOP instances for which all service points are reachable. In particular, we selected those instances with the smallest, largest, and median maximum route durations in the group, resulting in 43 instances as outlined in Table 2, where $\text{nInst}_{|\mathcal{K}|=2}$, $\text{nInst}_{|\mathcal{K}|=3}$, and $\text{nInst}_{|\mathcal{K}|=4}$ indicate the number of instances per group with two, three, and four vehicles.

Then, for each of these 43 instances, five TOPO instances are derived by applying a three-step procedure. In the first step, a service radius ρ defining the maximum distance between a service point and the consumers it can serve is computed as $\rho = 0.5 * \min\{t_{ij} : i,j \in \mathcal{S} : i \neq j\}$. The service radius ρ subsequently is used to define nonoverlapping circular service regions centered around each service point. In the second step, for every service point (within its service region), as many consumers as the profit of the associated service point in the original TOP instance are first randomly generated and then allocated. In the third step, we compute the smallest value of ρ such that $\sum_{i\in\mathcal{S}} |\mathcal{C}_i| \geq |\mathcal{C}| * (1+\gamma)$, where $\gamma = 0.1, 0.2, 0.3, 0.4$, and 0.5. This results into a total of 215 TOPO instances with varying degrees of overlap. This procedure guarantees that the optimal solution value of the original TOP instance is a valid lower bound to the optimal solution value of any of the resulting five TOPO instances, thus allowing a direct assessment of the impact of increasing overlaps among service regions. Further details on the instance generator are available upon request.

Table 1. Features of the Chao Instances

Family	$\lvert\mathcal{S}\rvert$	nInst	$T_{\lvert\mathcal{K}\rvert=2}$	$T_{\lvert\mathcal{K}\rvert=3}$	$T_{\lvert\mathcal{K}\rvert=4}$
1	30	54	2.5, 42.5	1.7, 28.3	1.2, 21.2
2	19	33	7.5, 22.5	5.0, 15.0	3.8, 11.2
3	31	60	7.5, 55.0	5.0, 36.7	3.8, 27.5
4	98	60	25.0, 120.0	16.7, 80.0	12.5, 60.0
5	64	78	2.5, 65.0	1.7, 43.3	1.2, 32.5
6	62	42	7.5, 40.0	5.0, 26.7	3.8, 20.0
7	100	60	10.0, 200.0	6.7, 133.3	5.0, 100.0

Table 2. Features of the Selected Chao Instances

Family	$\lvert\mathcal{S}\rvert$	nInst	$\text{nInst}_{\lvert\mathcal{K}\rvert=2}$	$\text{nInst}_{\lvert\mathcal{K}\rvert=3}$	$\text{nInst}_{\lvert\mathcal{K}\rvert=4}$
1	30	6	3	3	0
3	31	5	3	2	0
4	98	9	3	3	3
5	64	9	3	3	3
6	62	7	3	3	1
7	100	7	3	3	1

6.1.2. Preliminary Computational Results of the Exact Methods. To compare the performance of the three branch-and-cut-and-price algorithms, we first conducted preliminary experiments on all 215 TOPO instances with a short time limit of 900 seconds. In particular, to assess the impact of the implemented dynamic *ng*-path acceleration technique, the three algorithms were tested under two settings: static and dynamic, depending on whether the sets $\mathcal{N}_i$ are defined in a static or dynamic way. The dynamic setting is the default one. The algorithms under the static setting are obtained by imposing $\eta_{N_i} = 11$.

Tables 3 and 4 summarize these preliminary experiments by grouping them by degree of overlap and number of vehicles, respectively. For each group of instances, each table reports the number of instances (nInst), the number of instances solved to optimality (opt), and the average final percentage gap over the open instances (gap_{Op}) for each of the three algorithms under the two settings. The average percentage gap for each instance is computed as $(\frac{\text{ub}_\text{F}}{\text{blb}} - 1) * 100$, where ub_F is the final best upper bound and blb is the best lower bound computed.

Table 3 shows that, under the static setting, BCP1 solves more instances (i.e., 143) than Baseline and BCP2, which solve 104 and 141 instances, respectively,

Table 3. Preliminary Results, Grouped by Degree of Overlap, of `Baseline`, `BCP1`, and `BCP2` (Static and Dynamic)

		Static						Dynamic					
		`Baseline`		`BCP1`		`BCP2`		`Baseline`		`BCP1`		`BCP2`	
Overlap (%)	nInst	opt	gap_{Op}	opt	gap_{Op}	opt	gap_{Op}	opt	gap_{Op}	opt	gap_{Op}	opt	gap_{Op}
10	43	26	0.59	32	0.35	33	0.47	31	0.52	35	0.25	30	0.92
20	43	26	1.23	28	0.47	28	1.02	25	1.03	30	0.34	29	0.86
30	43	21	1.78	30	0.55	27	1.09	22	1.42	30	0.39	29	1.08
40	43	16	2.84	27	0.72	26	0.87	19	2.70	29	0.59	25	1.41
50	43	15	3.35	26	0.72	27	0.66	16	3.13	29	0.46	27	1.19
All	215	104	1.96	143	0.56	141	0.82	113	1.76	153	0.41	140	1.09

Table 4. Preliminary Results, Grouped by Number of Vehicles, of `Baseline`, `BCP1`, and `BCP2` (Static and Dynamic)

		Static						Dynamic					
		`Baseline`		`BCP1`		`BCP2`		`Baseline`		`BCP1`		`BCP2`	
$\|K\|$	nInst	opt	gap_{Op}	opt	gap_{Op}	opt	gap_{Op}	opt	gap_{Op}	opt	gap_{Op}	opt	gap_{Op}
2	90	49	1.96	62	0.62	58	1.24	51	1.81	68	0.38	55	1.94
3	85	38	1.99	55	0.58	56	0.55	44	1.72	58	0.44	58	0.54
4	40	17	1.87	26	0.39	27	0.47	18	1.73	27	0.38	27	0.34
All	215	104	1.96	143	0.56	141	0.82	113	1.76	153	0.41	140	1.09

and the final gap is, on average, smaller (i.e., 0.56% versus 1.96% and 0.82%, respectively). Similar results are achieved under the dynamic setting. The table also indicates that `Baseline` and `BCP1` solve more instances and provide lower gaps under the dynamic setting than the static setting, whereas `BCP2` has better performance under the static setting. Nevertheless, `BCP1` under both settings performs better than the other two algorithms. In particular, for `BCP2`, the increased computational overhead seems to have a substantial impact on its performance, nullifying the eventual improvement in the linear relaxation bound (see Section 4.2.2). We can also see that the performance of all six algorithms tends to be better when the overlaps are smaller (i.e., 10% and 20% overlap) and deteriorates when the overlaps are larger (i.e., 40% and 50% overlap). This indicates that instances with larger overlaps among service regions are more challenging. Table 4 indicates that `BCP1` is significantly better than `Baseline` and `BCP2` on instances with two vehicles.

A more detailed comparison of the performance of `BCP1` in the static (`BCP1-Static`) and dynamic (`BCP1-Dynamic`) settings is provided in Tables 5 and 6 on the basis of the 138 instances solved to optimality under both settings. Each row of these two table reports the number of instances commonly solved to optimality (opt) in the corresponding group, and, w.r.t. the instances solved, the average upper bound at the root node (ub_R), the average root solution time (cpu_R), and the average total solution time per instance (cpu_F). Average values for `BCP1-Dynamic` are reported as geometric means of the ratios with respect to the corresponding values of the static version. For `BCP1-Dynamic`, we also report the average cardinality of the sets $\mathcal{N}_i$. Tables 5 and 6 show that the versions of `BCP1` provide similar average upper bounds at the root node, even though the cardinality of the sets $\mathcal{N}_i$ is on average just 4.1 for `BCP1-Dynamic`. Although `BCP1-Dynamic` takes on average 65% more time than `BCP1-Static` to terminate the root node, it takes on average 13% less time to close the instances. Moreover, it seems that the number of vehicles does not affect the performance of the two algorithms.

6.1.3. Results of the Exact Method. As the preliminary results for a time limit of 900 seconds suggested that `BCP1` (in its dynamic setting, i.e., the default setting) is

Table 5. Full Comparison Between `BCP1-Static` and `BCP1-Dynamic`: Results Grouped by Degree of Overlap

		`BCP1-Static`			`BCP1-Dynamic`			
Overlap (%)	opt	ub_R	cpu_R	cpu_F	$\|N_i\|$	ub_R	cpu_R	cpu_F
10	32	312.2	6.0	123.7	4.2	1.00	1.50	0.85
20	27	311.5	3.0	62.7	4.0	1.00	1.55	1.00
30	28	289.7	2.8	101.2	4.1	1.01	1.94	0.82
40	26	297.2	2.3	119.9	4.0	1.00	1.84	0.94
50	25	252.4	2.4	114.3	4.1	1.00	1.50	0.75
All	138	293.8	3.4	104.8	4.1	1.00	1.65	0.87

Table 6. Full Comparison Between BCP1-Static and BCP1-Dynamic: Results Grouped by Number of Vehicles

		BCP1-Static			BCP1-Dynamic			
$\|K\|$	opt	ub_R	cpu_R	cpu_F	$\|N_i\|$	ub_R	cpu_R	cpu_F
2	59	320.7	3.3	123.7	4.2	1.00	1.41	0.93
3	53	247.3	2.9	86.3	4.2	1.00	2.28	0.97
4	26	327.7	4.6	99.4.9	3.6	1.00	1.22	0.60
All	138	293.8	3.4	104.8	4.1	1.00	1.65	0.87

on average superior to the other algorithms, we extensively tested it on all 215 instances for a time limit of one hour to solve each instance. Table 7 summarizes the results achieved grouped by degree of overlap and by number of vehicles. The following information is reported: number of instances (nInst), average gap at the root node before changing the $\mathcal{N}_i$ sets and before adding SR inequalities (gap_0), average final gap at the root node (gap_R), number of instances solved to optimality (opt), average solution time (cpu_F), number of instances open (nOpen), and average gap between the highest upper bound of the unexplored nodes left in the search tree and the best lower bound computed (gap_{Op}).

Table 7 shows that BCP1 can solve 179 of the 215 instances, and the average gap left for the 36 open instances is quite small (i.e., 0.44%). The average solution time is five minutes. When comparing the results on instances with different overlaps, it is clear that the higher the overlap, the more difficult the instances are: this is because the root node provides worse upper bounds when the overlaps increase.

Table 7. Summary of the Results of BCP1 with One Hour of the Time Limit

Overlap (%)	$\|K\|$	nInst	gap_0	gap_R	opt	cpu_F	nOpen	gap_{Op}
10		43	1.26	0.48	38	214.2	5	0.23
	2	18	1.17	0.42	17	233.3	1	0.00
	3	17	1.50	0.48	15	262.4	2	0.27
	4	8	0.96	0.62	6	39.6	2	0.32
20		43	1.34	0.52	38	353.2	5	0.39
	2	18	1.29	0.34	18	461.5	0	
	3	17	1.52	0.64	14	312.6	3	0.33
	4	8	1.07	0.70	6	122.8	2	0.48
30		43	1.48	0.68	36	303.1	7	0.41
	2	18	1.53	0.65	15	327.2	3	0.32
	3	17	1.67	0.74	15	368.7	2	0.45
	4	8	0.93	0.62	6	78.6	2	0.52
40		43	1.58	0.80	34	309.4	9	0.41
	2	18	1.57	0.69	15	283.4	3	0.39
	3	17	1.94	1.03	13	363.8	4	0.48
	4	8	0.82	0.57	6	256.6	2	0.31
50		43	1.66	0.84	33	319.9	10	0.59
	2	18	1.67	0.74	16	270.6	2	0.84
	3	17	2.12	1.16	11	252.8	6	0.58
	4	8	0.68	0.42	6	574.3	2	0.39
All		215	1.46	0.67	179	299.1	36	0.44

6.1.4. Results of LNS. Table 8 summarizes the computational results of LNS on the 215 synthetic TOPO instances. Results are grouped by degree of overlap and by number of vehicles. On each instance, LNS was run 10 times. Columns report the following information: number of instances (nInst), the number of instances on which LNS found the best-known solution in any of the runs even if it was not proved to be optimal by BCP1 (bk), the number of instances on which LNS found an optimal solution in any of the runs (opt), the percentage gap between the best solution found by the LNS and the best primal bound found by BCP1 (gap_{BPB}; notice that negative gaps mean that the best solution found by LNS is better than the best solution found by BCP1), the percentage gap between the best solution found by LNS and the best dual bound provided by the BCP1 (gap_{BDB}), the percentage gap between the average best solution found by LNS over all runs and the best dual bound provided by BCP1 (gap_{ADB}), and the average solution time (cpu_F). Table 8 shows that LNS could find the best-known solutions of 183 out of 215 instances. Moreover, the average gaps between the best and the average lower bounds provided by LNS and the upper bounds provided by BCP1 are small (i.e., 0.13% and 0.23%, respectively), yet these values may be overestimated because not all upper bounds provided by BCP1 were proved to be optimal. We can also observe that, similarly to BCP1, LNS performs better on instances with smaller overlaps.

We conducted further experiments, summarized in Table 9, to shed light on the effectiveness of some key components of the LNS presented in Section 5. In particular, we tested the impact of (i) the preprocessing step, (ii) the acceptance criterion of the intraroute operators, (iii) the *replaceRandom* procedure, and (iv) the selection criterion of the solution from the pool Ω. Therefore, we tested four additional versions of the LNS on all 215 synthetic TOPO instances. In LNS2, the preprocessing step is not performed, so the *replace* and *add* operators need to compute the profit of the solutions in the neighborhoods in a more computationally intensive way. In this version, a time limit of 720.0 seconds was imposed on each experiment. This time limit exceeds the maximum computation time (i.e., 658.1 seconds) observed when LNS is used. In LNS3, the acceptance criterion of the intraroute operators is not the minimization of the length of the shortest route, but rather the minimization of the length of all routes in the incumbent solution. In LNS4, the *replaceRandom* procedure is removed. Finally, whereas in LNS, in each iteration a solution is randomly selected from the pool Ω according to a uniform probability distribution, in LNS5, the probability is proportional to the solution quality.

Table 8. Summary of the Results of LNS on All 215 TOPO Instances

Overlap (%)	$\|K\|$	nInst	bk	opt	gap_{BPB}	gap_{BDB}	gap_{ADB}	cpu_F
10		43	36	33	–0.05	0.07	0.17	200.9
	2	18	14	14	–0.02	0.07	0.17	134.0
	3	17	15	14	–0.10	0.05	0.14	208.5
	4	8	7	5	–0.02	0.11	0.20	335.0
20		43	39	35	–0.04	0.07	0.15	205.9
	2	18	16	16	0.01	0.01	0.12	140.8
	3	17	16	14	–0.08	0.07	0.14	207.8
	4	8	7	5	–0.04	0.19	0.25	348.5
30		43	36	30	–0.04	0.15	0.23	206.2
	2	18	14	12	–0.06	0.12	0.18	145.2
	3	17	15	13	–0.07	0.12	0.21	199.0
	4	8	7	5	0.07	0.27	0.37	358.5
40		43	36	31	–0.03	0.15	0.28	205.2
	2	18	13	13	0.07	0.15	0.25	147.0
	3	17	16	13	–0.15	0.15	0.29	195.9
	4	8	7	5	0.01	0.17	0.33	355.9
50		43	36	29	–0.07	0.20	0.34	211.5
	2	18	14	13	0.00	0.18	0.33	151.1
	3	17	15	12	–0.12	0.26	0.40	199.1
	4	8	7	5	–0.16	0.12	0.27	374.0
All		215	183	158	–0.04	0.13	0.23	205.9

Table 9 shows that versions LNS2 and LNS4 have a significantly worse performance than LNS, thus proving that the preprocessing step and the *replaceRandom* procedure are crucial. The computational results for LNS3 illustrate that the alternative acceptance criterion of minimizing the length of all routes prevents finding two optimal and three best-known solutions found by LNS and results in an increased computing time. Finally, the results of LNS5 and LNS are nearly identical.

6.2. Computational Results on Benchmark TOP Instances

As the TOPO generalizes the TOP, which has been extensively studied in the literature, we tested BCP1 and LNS also on the Chao TOP instances to compare our solution method against the state-of-the-art solution methods from the literature. The results are summarized in this section.

6.2.1. Results of the Exact Method. Table 10 summarizes the computational results achieved by BCP1 on the 387 TOP instances. Results are grouped by family of instances and reported in the same columns as in Table 7. Table 10 shows that BCP1 can solve all but 16 instances to optimality, with an average solution time of 122.7 seconds. The success of BCP1 is clearly due to the quality of the upper bounds provided by the root node (i.e., 1.26% and 0.44% before and after adding SR inequalities and changing the sets $\mathcal{N}_i$). The gap left on the 16 open instances is also quite small (i.e., 0.30%). It is worth noting that, considering the best lower bounds available from the literature, BCP1 could prove the optimality of three additional instances, namely, p4.2.s, p4.3.t, and p4.4.r (see Table EC.11), in addition to the 371 instances solved.

In Table 11, we compare the results achieved by BCP1 with those achieved by the state-of-the-art solution methods from the literature, in particular with the branch-and-price algorithm of Boussier, Feillet, and Gendreau (2007) (BFG07), the branch-and-cut algorithm of Dang, El-Hajj, and Moukrim (2013) (DEM13), the branch-and-cut-and-price algorithm of Keshtkaran et al. (2016) (KZBV16), the branch-and-cut plane algorithm

Table 9. Evaluation of the Different Functionality Modules of LNS

Version	nInst	bk	opt	gap_{BPB}	gap_{BDB}	gap_{ADB}	cpu_F
LNS	215	183	158	–0.04	0.13	0.23	205.9
LNS2	215	123	110	0.32	0.43	0.86	572.7
LNS3	215	180	156	–0.05	0.12	0.22	241.2
LNS4	215	139	115	0.15	0.32	0.71	113.6
LNS5	215	183	158	–0.04	0.13	0.23	205.2

Table 10. Summary of the Results of BCP1 on the TOP Instances

Family	nInst	gap_0	gap_R	opt	cpu_F	nOpen	gap_{Op}
1	54	0.65	0.39	54	0.1	0	
2	33	0.30	0.13	33	0.0	0	
3	60	1.91	0.96	60	0.7	0	
4	60	1.23	0.56	49	509.0	11	0.26
5	78	1.00	0.34	75	137.0	3	0.47
6	42	0.34	0.31	42	12.0	0	
7	60	2.71	0.26	58	168.3	2	0.19
All	387	1.26	0.44	371	122.7	16	0.30

Table 11. Comparison Between BCP1 and the State-of-the-Art Exact Methods on TOP Instances

		BFG07 Intel Pentium IV (3.20 GHz)		DEM13 AMD Opteron (2.60 GHz)		KZBV16 Intel Core i7 (3.60 GHz)		EDM16 AMD Opteron (2.60 GHz)		BMS18 Intel Xeon (W3680)		BCP1 Intel Core (i7-6700U)	
Family	nInst	opt	cpu_F	opt	cpu_F	opt	cpu_F	opt	cpu_F	opt	cpu_F	opt	cpu_F
1	54	51	38.2	54	na	54	12.9	54	na	54	1.1	54	0.1
2	33	33	0.1	33	na	33	0.1	33	na	33	0.2	33	0.0
3	60	50	103.8	60	na	60	258.3	60	na	60	184.9	60	0.7
4	60	25	459.9	22	na	20	120.4	30	na	39	870.4	49	509.0
5	78	48	200.6	44	na	60	252.2	54	na	60	517.9	75	137.0
6	42	36	286.1	42	na	36	203.2	42	na	36	22.1	42	12.0
7	60	27	203.2	23	na	38	768.6	27	na	45	992.8	58	168.3
All	387	270		278		301		300		327		371	

of El-Hajj, Dang, and Moukrim (2016) (EDM16), and the branch-and-cut algorithm of Bianchessi, Mansini, and Speranza (2018) (BMS18). For the sake of completeness, we also report the processor details of the computing environments used by the six methods. Results are summarized by family of instances. For each method, we report the number of instances solved to optimality (opt) and the average solution time (cpu_F). We report "na" when the average solution time is not available.

Table 11 shows that BCP1 can solve more instances than all the methods available from the literature. In particular, BCP1 can solve 44 instances more than the recent branch-and-cut algorithm of Bianchessi, Mansini, and Speranza (2018) and 71 instances more than the most recent column-generation-based algorithm of Keshtkaran et al. (2016). We can also mention that BCP1 solves 33 out of the 49 instances that were still open (see Section 2.1 and the e-companion). Finally, we note that for instance p4.4.n (see Table EC.11), we computed an optimal value of 976, which is inconsistent with the primal bound of 977 reported in Tang and Miller-Hooks (2005) but consistent with all the other primal bounds reported in the literature so far.

6.2.2. Results of LNS.

Table 12 summarizes the computational results of LNS on the TOP instances and compares them with the results of Dang, Guibadj, and Moukrim (2013) (DGM13), Vidal et al. (2015) (VOP15), and Ke et al. (2016) (KZLC16); the reader is referred to Section 2 for a discussion of these methods. For the sake of completeness, we also report the processor details of the computing environments used to test the four solution methods. For each method, the table reports the number of instances on which the best-known solution was found (bk), the average gap between the best solution found by the heuristic and the best-known solution available (gap_{BPB}), and the average solution time (cpu_F). The VOP15 method was tested on only a subset of instances, so the last two rows of the table summarize the results over all seven families for the DGM13 and KZLC16 methods and LNS, and over Families 4–7 for VOP15 and LNS.

Table 12 shows that LNS is able to find the best-known solution of all but one TOP instance in a few minutes of computing time. Moreover, LNS is competitive with VOP15 on instances of Families 4–7 but cannot find one of the best-known solutions achieved by DGM13 and KZLC16.

Table 12. Comparison Between LNS and the State-of-the-Art Heuristic Methods on TOP Instances

		DGM13 AMD Opteron (2.60 GHz)			VOP15 Intel Xeon (3.07 GHz)			KZLC16 Intel Core i5 (3.2 GHz)			LNS Intel Core i7-6700U		
Family	nInst	bk	gap_{BPB}	cpu_F	bk	gap_{BPB}	cpu_F	bk	gap_{BPB}	cpu_F	bk	gap_{BPB}	cpu_F
1	54	54	0.0	2.1				54	0.0	6.7	54	0.0	3.2
2	33	33	0.0	0.4				33	0.0	1.4	33	0.0	0.3
3	60	60	0.0	3.2				60	0.0	9.6	60	0.0	4.2
4	60	60	0.0	214.1	58	0.0038	224.2	60	0.0	108.5	60	0.0	209.2
5	78	78	0.0	49.3	78	0.0	110.6	78	0.0	22.9	77	0.0043	45.3
6	42	42	0.0	43.5	42	0.0	54.2	42	0.0	26.9	42	0.0	44.7
7	60	60	0.0	96.8	60	0.0	166.1	60	0.0	54.3	60	0.0	114.1
1–7	387	387	0.0	58.5				387	0.0	32.9	386	0.0	60.1
4–7	240				238	0.0	138.8				239	0.0	103.3

6.3. Computational Results on Real-Life Instances

To illustrate the managerial relevance of the TOPO and the performance of the proposed solution approaches, real-life instances were provided by Geldmaat representing the cash replenishment problems they face in four major cities in the Netherlands (i.e., Almere, Amsterdam, Arnhem, and Tilburg). The service points represent ATMs that require replenishment and consumers represent the bank account holders that have to be served.

The instances of the four cities have the following numbers of ATMs/service points ($|\mathcal{S}|$) and consumers/bank account holders ($|\mathcal{C}|$): Almere, $|\mathcal{S}| = 32$ and $|\mathcal{C}| = 112,803$; Amsterdam, $|\mathcal{S}| = 100$ and $|\mathcal{C}| = 203,717$; Arnhem, $|\mathcal{S}| = 33$ and $|\mathcal{C}| = 74,805$; Tilburg, $|\mathcal{S}| = 35$ and $|\mathcal{C}| = 92,688$. For the cities of Almere, Arnhem, and Tilburg, we generated two instances: the first with one vehicle and the second with two vehicles. For the city of Amsterdam, we generated four instances, with one, two, three, and four vehicles, respectively. In total, we considered 10 instances. Each instance features a maximum route duration equal to 480 minutes, which corresponds to the typical working day of eight hours. Bank account holders (identified by the postal code of residence) are considered served if there exists a replenished ATM within five kilometers from the residence postal code.

6.3.1. Results of the Exact Method. Table 13 reports the results of BCP1 on the 10 real-life instances. A time limit of one hour was used for each instance. For each instance (identified by the city and the number of vehicles), the table reports the best lower bound (blb) found by either BCP1 or LNS, the upper bound at the root node (ub_R) and the corresponding gap (gap_R), the final upper bound (ub_F) and the corresponding gap (gap_F), the final lower bound (lb_F) and the corresponding gap (gap_{Op}) if the instance was not solved to optimality, the number of nodes of the search tree (nds), and the total solution time (cpu_F).

Table 13 shows that the six instances for the cities of Almere, Arnhem, and Tilburg and the two instances of Amsterdam with one and four vehicles can easily be solved to optimality. BCP1 cannot solve the Amsterdam instances with two and three vehicles, but managed to find upper bounds that are 0.45% and 0.21% from the best-known lower bound. We can also notice that the gap at the root node is never larger than 0.48%. Notice that all bank account holders can be served in Almere, Arnhem, and Tilburg with two vehicles, and in Amsterdam with four vehicles.

6.3.2. Results of the LNS. Table 14 summarizes the computational results of LNS on the 10 real-life instances. The metaheuristic used the same parameter setting as in the experiments for the synthetic instances. Similarly, 10 runs per instance were performed. For each instance, the table reports the worst (lb_W), average (lb_A), and best (lb_B) lower bounds found along with their corresponding gaps computed with respect to the best-known upper bound available (gap_{WDB}, gap_{ADB}, gap_{BDB}), the gap between lb_B and the best lower bound lb_F found by BCP1 (gap_{BPB}), the best lower bound computed (blb), and the average solution time (cpu_F) over the 10 runs.

We can observe that LNS manages to find solutions that are within 0.74% from the best-known upper bound provided by BCP1 on all instances. Moreover, LNS can improve the final lower bound provided by BCP1 on the Amsterdam instances with two and three vehicles.

6.3.3. Managerial Implications. To illustrate the managerial relevance of our research, we conducted a further computational study to assess the impact of solving the 10 real-life TOPO instances as TOP instances. We consider two intuitive approaches of solving TOPO instances as TOP instances. The first approach (hereafter, TOP1) solves a TOP instance derived from the original TOPO instance where the profit of each service point/ATM is equal to the number of consumers/bank account holders that are within its service region. Customers in overlapping zones can therefore be assigned to multiple service points. The second approach (hereafter, TOP2) solves a TOP instance where

Table 13. Results of BCP1 on Real-Life Instances

City	$\|K\|$	blb	ub_R	gap_R	ub_F	gap_F	lb_F	gap_{Op}	nds	cpu_F
Almere	1	103,173	103,271	0.09	103,173	0.00	103,173		22	190.3
Almere	2	112,803	112,803	0.00	112,803	0.00	112,803		4	0.6
Amsterdam	1	116,611	116,611	0.00	116,611	0.00	116,611		1	146.0
Amsterdam	2	182,779	183,648	0.48	183,593	0.45	172,457	5.65	30	Time limit
Amsterdam	3	202,175	202,606	0.21	202,605	0.21	199,345	1.40	2	Time limit
Amsterdam	4	203,717	203,717	0.00	203,717	0.00	203,717		39	231.5
Arnhem	1	74,643	74,643	0.00	74,643	0.00	74,643		2	35.2
Arnhem	2	74,805	74,805	0.00	74,805	0.00	74,805		10	2.2
Tilburg	1	73,522	73,522	0.00	73,522	0.00	73,522		1	2.0
Tilburg	2	92,688	92,688	0.00	92,688	0.00	92,688		1	0.2

Table 14. Results of LNS on Real-Life Instances

City	$\|K\|$	lb_W	gap_{WDB}	lb_A	gap_{ADB}	lb_B	gap_{BDB}	gap_{BPB}	blb	cpu_F
Almere	1	101.919	1.33	102,785.9	0.47	103,097	0.17	0.07	103,173	26.8
Almere	2	112,803	0.00	112,803.0	0.00	112,803	0.00	0.00	112,803	0.0
Amsterdam	1	113,795	2.47	113,994.3	2.30	115,788	0.71	0.71	116,611	155.5
Amsterdam	2	181,431	1.22	181,800.7	1.02	182,779	0.48	–5.99	182,779	613.7
Amsterdam	3	201,947	0.33	202,068.5	0.27	202,175	0.21	–1.42	202,175	1,435.1
Amsterdam	4	202,469	0.62	202,812.8	0.45	203,182	0.26	0.26	203,717	1,207.1
Arnhem	1	74,480	0.22	74,563.3	0.11	74,635	0.01	0.01	74,643	15.2
Arnhem	2	74,805	0.00	74,805.0	0.00	74,805	0.00	0.00	74,805	0.0
Tilburg	1	71,146	3.34	72,172.2	1.87	72,980	0.74	0.74	73,522	10.0
Tilburg	2	91,220	1.61	92,033.1	0.71	92,625	0.07	0.07	92,688	24.2

each consumer/bank account holder is a priori assigned to its closest ATM, and the profit of each ATM is then the number of bank account holders for which that ATM is the closest one. Therefore, TOP1 overestimates the profit of each ATM, and TOP2 underestimates it. For the solution of both TOP1 and TOP2, the profit that will be collected in reality is obtained by a postprocessing procedure.

Table 15 reports, for each of the 10 real-life instances and each of the approaches (TOP1, TOP2), the real value of the best solution found (blb) and its percentage deviation (Δ) w.r.t. the value of the best-known solution found solving directly the instance as a TOPO instance (TOPO blb). For all cases, solutions are computed using LNS.

The most striking observation from Table 15 is that, on average, the solutions obtained by TOP1 and TOP2 serve respectively 11.8% and 16.8% fewer bank account holders than the solutions found by considering the problem as a TOPO. In other words, wrongfully assessing a TOPO as a TOP can lead to significantly lower profits or service levels in real-life applications.

7. Conclusions and Future Research

Motivated by a real-life ATM cash replenishment problem encountered in the Netherlands, we have investigated a new generalization of the team orienteering problem, called the TOP with overlaps. We have proposed exact methods based on column generation for the exact solution of the problem, where our main contribution has been in exploiting the *ng*-path relaxation to obtain high-quality dual bounds for the problem. We have also proposed a metaheuristic based on large neighborhood search. The performance of the proposed solution methods has been assessed by an extensive computational study on synthetic and real-life instances. The computational results have shown that the proposed methods can find high-quality bounds of all considered instances. We have also shown that the proposed solution methods are competitive with the state-of-the-art solution methods for the TOP. In particular, we could find the optimal solution of 96% of the well-known Chao instances and close 33 open instances. From a managerial standpoint, we have also shown that modeling a real-life cash replenishment problem as a TOPO provides significantly better solutions than solving it as a TOP and can help to strongly improve service levels.

Table 15. Comparison of the Performance of TOPO, TOP1, and TOP2 on Real-Life Instances

		TOPO	TOP1		TOP2	
City	$\|K\|$	blb	blb	Δ	blb	Δ
Almere	1	103,097	78,900	–23.5	84,735	–17.8
Almere	2	112,803	112,803	0.0	112,803	0.0
Amsterdam	1	115,788	90,110	–22.2	64,421	–44.4
Amsterdam	2	182,779	135,014	–26.1	109,805	–39.9
Amsterdam	3	202,175	173,114	–14.4	147,575	–27.0
Amsterdam	4	203,182	192,970	–5.0	183,633	–9.6
Arnhem	1	74,635	65,374	–12.4	64,846	–13.1
Arnhem	2	74,805	74,805	0.0	74,805	0.0
Tilburg	1	72,980	63,061	–13.6	62,469	–14.4
Tilburg	2	92,625	91,787	–0.9	91,175	–1.6
Average				–11.8		–16.8

Acknowledgments

The authors are grateful to the associate editor and to the anonymous referees for their comments that improved the presentation of this paper. The authors would also like to thank Geldmaat for providing the real-life instances used in the computational study.

References

Adulyasak Y, Cordeau JF, Jans R (2012) Optimization-based adaptive large neighborhood search for the production routing problem. *Transportation Sci.* 48(1):20–45.

Archetti C, Hertz A, Speranza MG (2007) Metaheuristics for the team orienteering problem. *J. Heuristics* 13(1):49–76.

Archetti C, Speranza MG, Vigo D (2014) Vehicle routing problems with profits. Toth P, Vigo D, eds. *Vehicle Routing: Problems, Methods, and Applications*, 2nd ed. (SIAM, Philadelphia), 273–297.

Archetti C, Feillet D, Hertz A, Speranza MG (2009) The capacitated team orienteering and profitable tour problems. *J. Oper. Res. Soc.* 60(6):831–842.

Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.* 59(5):1269–1283.

Bianchessi N, Mansini R, Speranza MG (2018) A branch-and-cut algorithm for the team orienteering problem. *Internat. Trans. Oper. Res.* 25(2):627–635.

Bouly H, Dang DC, Moukrim A (2010) A memetic algorithm for the team orienteering problem. *4OR* 8(1):49–70.

Boussier S, Feillet D, Gendreau M (2007) An exact algorithm for team orienteering problems. *4OR* 5(3):211–230.

Butt SE, Cavalier TM (1994) A heuristic for the multiple tour maximum collection problem. *Comput. Oper. Res.* 21(1):101–111.

Chao IM, Golden BL, Wasil EA (1996) The team orienteering problem. *Eur. J. Oper. Res.* 88(3):464–474.

Dang DC, El-Hajj R, Moukrim A (2013) A branch-and-cut algorithm for solving the team orienteering problem. Gomes C, Sellmann M, eds. *Internat. Conf. AI OR Techniques Constraint Programming Combin. Optim. Problems,* Lecture Notes in Computer Science, vol. 7874 (Springer, Berlin), 332–339.

Dang DC, Guibadj RN, Moukrim A (2011) A PSO-based memetic algorithm for the team orienteering problem. Di Chio C, Brabazon A, Di Caro GA, Drechsler R, Farooq M, Grahl J, Greenfield G, Prins C, Romero J, Squillero G, et al., eds. *Applications of Evolutionary Computation* (Springer, Berlin), 471–480.

Dang DC, Guibadj RN, Moukrim A (2013) An effective PSO-inspired algorithm for the team orienteering problem. *Eur. J. Oper. Res.* 229(2):332–344.

Drexl M (2007) *On Some Generalized Routing Problems*. Dissertation, RWTH Aachen University, Aachen, Germany.

Dror M (1994) Note on the complexity of the shortest path models for column generation in VRPTW. *Oper. Res.* 42(5):977–978.

El-Hajj R, Dang DC, Moukrim A (2016) Solving the team orienteering problem with cutting planes. *Comput. Oper. Res.* 74(October):21–30.

Emde S, Schneider M (2018) Just-in-time vehicle routing for in-house part feeding to assembly lines. *Transportation Sci.* 52(3):657–672.

Gunawan A, Lau HC, Vansteenwegen P (2016) Orienteering problem: A survey of recent variants, solution approaches and applications. *Eur. J. Oper. Res.* 255(2):315–332.

Hübner A, Ostermeier M (2018) A multi-compartment vehicle routing problem with loading and unloading costs. *Transportation Sci.* 53(1):282–300.

Jepsen M, Petersen B, Spoorendonk S, Pisinger D (2008) Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.* 56(2):497–511.

Joncour C, Michel S, Sadykov R, Sverdlov D, Vanderbeck F (2010) Column generation based primal heuristics. *Electron. Notes Discrete Math.* 36(August):695–702.

Ke L, Archetti C, Feng Z (2008) Ants can solve the team orienteering problem. *Comput. Indust. Engrg.* 54(3):648–665.

Ke L, Zhai L, Li J, Chan FT (2016) Pareto mimic algorithm: An approach to the team orienteering problem. *Omega* 61(June):155–166.

Keshtkaran M, Ziarati K, Bettinelli A, Vigo D (2016) Enhanced exact solution methods for the team orienteering problem. *Internat. J. Production Res.* 54(2):591–601.

Kim BI, Li H, Johnson AL (2013) An augmented large neighborhood search method for solving the team orienteering problem. *Expert Systems Appl.* 40(8):3065–3072.

Lin SW (2013) Solving the team orienteering problem using effective multi-start simulated annealing. *Appl. Soft Comput.* 13(2): 1064–1073.

Masson R, Lehuédé F, Péton O (2013) An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Sci.* 47(3):344–355.

Pisinger D, Ropke S (2019) Large neighborhood search. Gendreau M, Potvin J-Y, eds. *Handbook of Metaheuristics,* International Series in Operations Research & Management Science, vol. 272 (Springer International Publishing, Cham), 99–127.

Poggi M, Viana H, Uchoa E (2010) The team orienteering problem: Formulations and branch-cut and price. Erlebach T, Lübbecke M, eds. *10th Workshop Algorithmic Approaches Transportation Model., Optim., Systems* (Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany), 142–155.

Righini G, Salani M (2006) Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optim.* 3(3):255–273.

Roberti R, Mingozzi A (2014) Dynamic ng-path relaxation for the delivery man problem. *Transportation Sci.* 48(3):413–424.

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Sci.* 40(4):455–472.

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. Maher M, Puget J-F, eds. *Internat. Conf. Principles Practice Constraint Programming* (Springer, Berlin), 417–431.

Solomon MM (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* 35(2):254–265.

Souffriau W, Vansteenwegen P, Berghe GV, Van Oudheusden D (2010) A path relinking approach for the team orienteering problem. *Comput. Oper. Res.* 37(11):1853–1859.

Tang H, Miller-Hooks E (2005) A tabu search heuristic for the team orienteering problem. *Comput. Oper. Res.* 32(6):1379–1407.

Vansteenwegen P, Van Oudheusden D (2007) The mobile tourist guide: An or opportunity. *OR Insight* 20(3):21–27.

Vansteenwegen P, Souffriau W, Van Oudheusden D (2011) The orienteering problem: A survey. *Eur. J. Oper. Res.* 209(1):1–10.

Vansteenwegen P, Souffriau W, Berghe GV, Van Oudheusden D (2009) A guided local search metaheuristic for the team orienteering problem. *Eur. J. Oper. Res.* 196(1):118–127.

Vidal T, Maculan N, Ochi LS, Vaz Penna PH (2015) Large neighborhoods with implicit customer selection for vehicle routing problems with profits. *Transportation Sci.* 50(2):720–734.